CMSC 417 Programming Assignment #2
Due September 24, 2001 (10:00 AM)

Introduction

The term project will require use of the Pthreads thread programming model. Like the socket interface, the interface to this library is rather complicated and confusing. Also, if you have never written a program with synchronization, you might find this concept difficult at first. In this project, you will create a queue abstraction that allows multiple threads to share information by placing items into queues and having other threads remove them. The background reading for this project is the Pthreads book. Chapters 1, 3 (up to pg. 97) , and 6 are worth reading.

The Assignment

You will implement a queue abstraction that has the following functions. This assignment is to be written in the C programming language. You should define a queue type and pass it as the first parameter to each of the functions. A skeleton of the prototypes for the queue abstraction are available from the class web page.

`Queue *createQueue(int maxItems)`
Allocate and initialize a new queue that can hold at most maxItems. If you are writing in C++, this should be the constructor function for the Queue class.

`void enqueue(Queue *q, void *item)`
Add item to the end of the queue. If the queue already contains maxItems, the enqueue operations should block until there is room in the queue.

`void *dequeue(Queue *q, int timeout)`
Remove item from the qeueue, and **block** (not busy wait) the calling thread if the queue is empty. `Timeout` specifies the amount of time (in microseconds) that the dequeue operation should wait when the queue is empty before returning. If `timeout` is negative, dequeue will never return unless there is an item in the queue. When the dequeue function returns due to a timeout, it should return `NULL`.

`void *head(Queue *q)`
Return head of queue without removing the item. This function should return NULL if the queue is empty.

`bool probe(Queue *q)`
Test if the queue contains an item. It will return true if the queue is non-empty.

Implementation Requirements

The entire implementation should be in the file queue.c. You **can only add fields** to the header file we supply, you may **not** change the existing functions' parameters or return types.

Like the first program, you should submit a tar file. The tar file should include a Makefile that compiles your queue implementation, a second file driver.c that will contain the function main, the header file, and the typescript output of running your version of the driver.

The version of driver that you write should create two new threads and one queue. The first thread should enqueue the numbers 1..10 into the queue, and the second thread should remove the items and print them out as they are removed. When you create the queue, it should have a maximum of 5 items. You should also test the timeout and other features of your program (our test cases will!). When we test your program, we will substitute our own version of driver.c with additional test cases.